Hopefully Interesting Git Topics

5/19/2010 8:45

Brent Perschbacher

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Cherry-picking Commits

- **What is cherry-picking?**
  - **Cherry-picking allows you to pull over a single commit from another branch and apply it to the current branch. This differs from a merge in that the merge pulls over all commits since the divergence point.**
- **Why cherry-pick?**
  - **Cherry-picking makes applying changes such as bug fixes to multiple branches fast and easy.**
- **Cherry-picking multiple commits.**
  - **You need to cherry-pick in the proper order. Each commit is a state of the files involved so rearranging commits will cause conflicts.**

How to Cherry-pick

It is as easy as:

git cherry-pick <commit id>

While on the branch you want to cherry-pick to. Assuming there are no conflicts the commit will be applied to the current branch with the same commit message. All that remains is to test and push.

Merging vs. Rebasing

- What is merging?
  - When to merge.
  - How to merge.
- What is rebasing?
- When to use one over the other.

What is Merging?

Merging is a way to get two diverged trees combined again.

Merging in git will create an additional commit that inherits from two other commits. This Merge commit will be the parent of any future commits.

When to Merge

You will want to merge if you have been working on a topic branch and are ready to pull your changes over to master, or a release branch.

Keep in mind that merging branches will merge all commits from the source branch to the target branch since the point of divergence.

How to Merge

Merging is fairly simple in git. From the branch you want to merge onto (target) simply use:

git merge <source branch>

What is Rebasing?

Rebasing can also be used to combine two diverged trees. However, it is more than that.

Rebasing can be used to reorder commits or to change commits. Using interactive rebasing you can go back and rewrite your history including update the files, add new files, change commit messages and/or drop commits completely. (these topics are outside the scope of this discussion)

Please don’t do that to commits that have been made public (pushed to SSG)

Rebasing Similar to a Merge

Rebasing can be used to “merge” two branches in a similar way to merge, however rebasing doesn’t create an extra commit. This is what “git pull --rebase” does. This can also be used to keep a topic branch up to date with master or another branch.

From the branch you want to update type:

git rebase <source branch>

This will roll back your commits since the divergence point, apply all the commits from the source branch since the divergence point and then re-apply your changes on top of those commits.

Replicating git pull --rebase

An easy example of rebasing against an existing branch is to replicate what git pull --rebase does. A pull is basically a shortcut for a fetch and a merge, when you use --rebase it makes pull a fetch and a rebase.

Assuming we are on master we would do:

git fetch

git rebase origin/master

Why origin/master? Because the fetch stores all the new information in the local copy of origin. Your master is a tracking branch of that copy.

How to Handle Conflicts

Conflicts in git aren't really different than cvs. You will get the same conflict markers in your files as you did with cvs. And you will need to manually fix the issues.

What is different in git is that you will need to tell git that you have fixed the issues and potentially to continue with what it was doing.

Conflicts from pulling or merging

If you get a conflict from a pull you will need to fix the conflicts manually. Once that is done you can tell git that you are finished fixing things with:

git add <file names>

git commit

The commit message will be populated with information about the conflict. You can edit it as you see fit.

This will create a merge commit.

Conflicts from Rebasing

This includes conflicts from “git pull --rebase”

Fix the conflicts as always then:

git add <file names>

git rebase --continue

There will be no merge commit and the commit message will be that of the commit that had the conflicts.

Undoing Commits

How we undo a commit depends on whether or not the commit has been made public yet. Both situations are pretty simple to handle though.

Sandia National Laboratories

Undoing Private Commits

Getting rid of a commit that you no longer need is relatively easy. If it is the last commit all you need to do is:

git reset --hard HEAD^

This will reset the current branch and working directory to the previous commit.

If you want to remove a commit that isn't the last commit you need to use interactive rebasing.

git rebase -i <commit id before commit in question>

Your editor will pop up and you will be presented with a list of commits with "pick" in front of them. All you need to do is find the commit you want to get rid of and remove that line. (much more can be done with interactive rebasing, but we won't get into that here)

Undoing Public Commits

Once a commit is public you cannot use the previous tricks to undo it. However, it isn't harder to undo public commits. You simply need to revert them.

git revert <commit id>

You will be prompted with your editor with a pre-made commit message that you can edit as you see fit. The message will already state that it is reverting the given commit.

Git vs eg

- What is eg?
- Why use eg?
- Some important differences between eg and git.

What is eg?

eg is a wrapper script written by Elijah Newren to make using git easier for people familiar with svn and cvs.

Why Use eg?

eg really is just another interface to git. It doesn't replace git and it isn't weaker than git. What it does do is change some of git's defaults to be more intuitive for people familiar with svn/cvs.

There are also some places where eg has different keywords that are more intuitive.

Important differences

eg has a couple important differences from git that you should be aware of.

eg push by default only pushes the branch you are currently on.

eg commit by default assumes that unstaged files are to be committed.

eg clone by default will set up tracking branches for all branches on origin.

eg's help will tell you of any differences between eg and git